R documentation

of ‘plot_compare_joint.Rd’ etc.

July 16, 2024

`plot_compare_joint` *Plot and compare joint posterior distribution(s) from ABC-SMC-(D)RF result*

Description

‘plot_compare_joint()‘ plots the joint posterior distribution(s) for the provided ABC-SMC-(D)RF result. It can also compare the joint posterior distributions for the provided ABC-SMC-(D)RF result with several ABC-SMC-(D)RF plots results.

Usage

```
plot_compare_joint(
  plots = NULL,
  abc_results,
  parameters_truth = NULL,
  parameters_labels = NULL,
  lims = NULL,
  nBins = 5
)
```

Arguments

`plots`	An existed ABC-SMC-(D)RF joint plots result. If provided, plot_compare_joint() will plot the new ABC-SMC-(D)RF result and compare it with provided plots results. If plots = NULL, plot_compare_joint() will make a new plot for the ABC-SMC-(D)RF result.
`abc_results`	An ABC-SMC-(D)RF result. Will be plotted out by the function.
`parameters_truth`	A dataframe containing true values of parameters from the ground-truth distributions. If provided, the function will plot the true values or distributions of parameters.
`parameters_labels`	A dataframe containing labels in the plots for corresponding parameters. If provided, parameter labels will be exhibited on the plots’ axes.
`lims`	A dataframe containing the maximum and minimum bounds for parameters. If provided, x-axis and y-axis will be scaled by them.
`nBins`	Number of contour bins shown in the plot. Default is 5.

Value

An ABC-SMC-(D)RF joint plots object ‘plots‘ containing the joint plots results of posterior distributions. The user can use the function to compare ABC-SMC-(D)RF joint plots with the joint posterior distribution(s) of other ABC-SMC-(D)RF result(s).

`plot_compare_qqplot` *Plot and compare marginal quantile-quantile plots from ABC-SMC-(D)RF result*

Description

‘plot_compare_qqplot()‘ plots the marginal quantile-quantile plots for inferred parameters and parameters from ground-truth distributions.

Usage

```
plot_compare_qqplot(
  plots = NULL,
  abc_results,
  parameters_truth,
  parameters_labels = NULL,
  lims = NULL
)
```

Arguments

`plots`	An existed ABC-SMC-(D)RF quantile-quantile plots result. If provided, plot_compare_qqplot() will plot the quantile-qantile plot for inferred parameters in the new ABC-SMC-(D)RF result and compare it with provided quantile-quantile plots result. If plots = NULL, plot_compare_qqplot() will make a new quantile-quantile plot for inferred parameters in the ABC-SMC-(D)RF result and true parameters.
`abc_results`	An ABC-SMC-(D)RF result. The function will plot the quantile-quantile plot between the inferred parameters from the ABC-SMC-(D)RF result and true parameters.
`parameters_truth`	A dataframe containing true values of parameters from the ground-truth distributions.
`parameters_labels`	A dataframe containing labels in the plots for corresponding parameters. If provided, parameter labels will be exhibited on the plots’ axes.
`lims`	A dataframe containing the maximum and minimum bounds for parameters. If provided, x-axis and y-axis will be scaled by them.

Value

An ABC-SMC-(D)RF quantile-quantile plots object ‘plots‘ containing the quantile-quantile plots results. The user can use the function to compare ABC-SMC-(D)RF quantile-quantile plots with the quantile-quantile plots of other ABC-SMC-(D)RF result(s).

plot_smcrf_marginal — *Plot distribution(s) in each iteration from ABC-SMC-(D)RF result*

Description

‘plot_smcrf_marginal()‘ plot the marginal distribution(s) for each iteration from an Approximate Bayesian Computation sequential Monte Carlo via random forest result.

Usage

```
plot_smcrf_marginal(
  smcrf_results,
  parameters_truth = NULL,
  parameters_labels = NULL,
  statistics_labels = NULL,
  plot_statistics = FALSE,
  xlimit = NULL,
  alpha = 0.3,
  plot_hist = FALSE
)
```

Arguments

`smcrf_results`	An ABC-SMC-(D)RF result containing the inference distributions of parameters from each iteration.
`parameters_truth`	A dataframe containing true values of parameters from the ground-truth distributions. If provided, the function will plot the true values or distributions of parameters.
`parameters_labels`	A dataframe containing labels in the plots for corresponding parameters. If provided, parameter labels will be exhibited on the plots’ axes.
`statistics_labels`	A dataframe containing labels in the plots for corresponding statistics. If provided, statistics labels will exhibit on the plots’ axes.
`plot_statistics`	A logic variable (plot_statistics = FALSE by default). If plot_statistics = TRUE, the marginal distributions in each iteration for corresponding statistics will also be output.
`xlimit`	A dataframe containing the maximum and minimum bounds for parameters. If provided, the x-axis will be scaled by them.
`alpha`	The numeric number to modify transparency. Default is 0.3.
`plot_hist`	A logic variable (plot_hist = FALSE by default). If plot_hist = TRUE, marginal distributions will be plotted in histograms.

See Also

`smcrf`

Examples

```
#    Dataframe containing the true parameters
parameters_truth <- data.frame(
    theta = 2
)
#    Dataframe containing the parameter labels
parameters_labels <- data.frame(
    parameter = c("theta"),
    label = c(deparse(expression(theta)))
)
#    Dataframe containing the x-axis bounds
xlimit <- data.frame(
    parameter = c("theta"),
    min = c(1),
    max = c(20)
)
```

---

plot_compare_marginal *Plot and compare marginal posterior distribution(s) from ABC-SMC-(D)RF result*

Description

'plot_compare_marginal()' plots the marginal posterior distribution(s) for the provided ABC-SMC-(D)RF result. It can also compare the marginal posterior distributions for the provided ABC-SMC-(D)RF result with several ABC-SMC-(D)RF plots results.

Usage

```
plot_compare_marginal(
  plots = NULL,
  abc_results,
  parameters_truth = NULL,
  parameters_labels = NULL,
  statistics_labels = NULL,
  xlimit = NULL,
  plot_statistics = FALSE,
  plot_hist = FALSE,
  plot_hist_point = FALSE,
  alpha = 0.3,
  plot_prior = FALSE
)
```

Arguments

`plots`	An existed ABC-SMC-(D)RF marginal plots result. If provided, plot_compare_marginal() will plot the new ABC-SMC-(D)RF result and compare it with provided plots results. If plots = NULL, plot_compare_marginal() will make a new plot for the ABC-SMC-(D)RF result.
`abc_results`	An ABC-SMC-(D)RF result. Will be plotted out by the function.

`parameters_truth`	A dataframe containing true values of parameters from the ground-truth distributions. If provided, the function will plot the true values or distributions of parameters.
`parameters_labels`	A dataframe containing labels in the plots for corresponding parameters. If provided, parameter labels will be exhibited on the plots' axes.
`statistics_labels`	A dataframe containing labels in the plots for corresponding statistics. If provided, statistics labels will exhibit on the plots' axes.
`xlimit`	A dataframe containing the maximum and minimum bounds for parameters. If provided, the x-axis will be scaled by them.
`plot_statistics`	A logic variable (plot_statistics = FALSE by default). If plot_statistics = TRUE, the marginal distributions in each iteration for corresponding statistics will also be output.
`plot_hist`	A logic variable (plot_hist = FALSE by default). If plot_hist = TRUE, marginal distributions will be plotted in histograms.
`plot_hist_point`	A logic variable (plot_hist_point = FALSE by default). If plot_hist_point = TRUE, marginal distributions will be plotted in histograms with points in the middle.
`alpha`	The numeric number to modify transparency. Default is 0.3.
`plot_prior`	A logic variable (plot_prior = FALSE by default) If plot_prior = TRUE, the prior distribution will be plotted out.

Value

An ABC-SMC-(D)RF marginal plots object 'plots' containing the marginal plots results of posterior distributions. The user can use the function to compare ABC-SMC-(D)RF marginal plots with the marginal posterior distribution(s) of other ABC-SMC-(D)RF result(s).

See Also

`smcrf`

`plot_smcrf_joint` *Plot joint distribution(s) of each iteration from ABC-SMC-(D)RF result*

Description

'plot_smcrf_joint()' plots the joint distribution(s) of two parameters for each iteration from an Approximate Bayesian Computation sequential Monte Carlo via random forest result.

Usage

```
plot_smcrf_joint(
  smcrf_results,
  parameters_truth = NULL,
  parameters_labels = NULL,
  lims = NULL,
  nBins = 5
)
```

Arguments

`smcrf_results`	An ABC-SMC-(D)RF result containing the inference distributions of parameters from each iteration.
`parameters_truth`	A dataframe containing true values of parameters from the ground-truth distributions. If provided, the function will plot the true values or distributions of parameters.
`parameters_labels`	A dataframe containing labels in the plots for corresponding parameters. If provided, parameter labels will be exhibited on the plots' axes.
`lims`	A dataframe containing the maximum and minimum bounds for parameters. If provided, x-axis and y-axis will be scaled by them.
`nBins`	Number of contour bins shown in the plot. Default is 5.

See Also

`smcrf`

Examples

```
lims <- data.frame(
    parameter = c("theta", "mu"),
    min = c(0, 0),
    max = c(10, 10)
)
```

`smcrf` *Approximate Bayesian Computation sequential Monte Carlo via random forests*

Description

'smcrf()' uses random forests to find the posterior distribution(s) for one or more parameters in a model. It implements the sequential Monte Carlo framework, where each iteration uses either ABC-RF (functions 'regAbcrf' and 'predict' in R package 'abcrf') or ABC-DRF (functions 'drf' and 'predict' in R package 'drf') to update the posterior distribution(s).

Usage

```
smcrf(
  method = "smcrf-single-param",
  statistics_target = NULL,
  smcrf_results = NULL,
  model,
  perturb,
  bounds = NULL,
  parameters_initial = NULL,
  nParticles,
  parallel = FALSE,
  n_cores = NULL,
  ...
)
```

Arguments

`method`	Random forest method to implement in each iteration ("smcrf-single-param" by default). method = "smcrf-single-param": implements ABC-RF for each parameter and results in their marginal posterior distributions. method = "smcrf-multi-param": implements ABC-DRF for all parameters and results in the joint posterior distribution.
`statistics_target`	A dataframe containing statistics from data. Column names are the statistics IDs. ‘smcrf()‘ only supports one row of statistics. If there are multiple observations, we recommend applying ‘smcrf()‘ to each row individually.
`smcrf_results`	An existing ABC-SMC-RF result. If provided, smcrf will continue ABC-SMC-RF from the last iteration of the previous run.
`model`	Model for the statistics. The function must take two inputs: a data frame parameters and logic variable parallel. The model will output a reference table. Each row contains parameters for each simulation and corresponding statistics.
`perturb`	A choice of kernel function that perturbs parameters for ABC-SMC-RF in each iteration. If perturb is a specified perturbation kernel function, each parameter follows the perturb function.
`bounds`	A dataframe containing bounds for each parameter. Usually no larger than the bounds of prior distribution.
`parameters_initial`	A dataframe containing the initial guess for parameters. Each column represents the prior distribution for corresponding parameter.
`nParticles`	A list of numbers showing the particles of ABC-SMC-RF. Each entry indicates the number of simulations in the corresponding iteration.
`parallel`	A logic variable (parallel = FALSE by default). If parallel = TRUE, the ABC-RF functions will be computed in parallel.
`n_cores`	Number of cores in used in parallel computation. When default with n_cores = NULL, the parallel function will use maximum number of available cores.
`...`	Additional arguments to be passed to ‘abcrf‘ or ‘drf‘.

Value

An object ‘smcrf_results‘ containing the results of the inference. If the posterior distributions have not converged to a satisfactory level, the user may continue with ‘smcrf(smcrf_results = smcrf_results, ...)‘, in which case ABC-SMC-(D)RF will continue iterating from the last run in ‘smcrf_results‘.

Examples

```
library(abcsmcrf)
#--------------------------------------------------------------------
#--------------------------ABC-SMC-RF for a model with one parameter
#--------------------------------------------------------------------
# Data to be fitted consists of two statistics s1 and s2
statistics_target <- data.frame(s1 = 0, s2 = 2)
# We then define a parametrized model for the statistics
model <- function(parameters) {
    statistics <- data.frame(
        s1 = parameters$theta - 1 + runif(nrow(parameters), -0.1, 0.1),
        s2 = parameters$theta + 1 + runif(nrow(parameters), -0.1, 0.1)
    )
    cbind(parameters, statistics)
}
# and a function to perturb the parameters with random noise between iterations
perturb <- function(parameters) {
   parameters$theta <- parameters$theta + runif(nrow(parameters), min = -0.1, max = 0.1)
    return(parameters)
}
# We start from initial guesses for theta from Uniform(-10, 10)
parameters_initial <- data.frame(theta = runif(100000, -10, 10))
# while ensuring that theta stays within bounds
bounds <- data.frame(
    parameter = c("theta"),
    min = c(-10),
    max = c(10)
)
# Finally, we run ABC-SMC-RF with 2 iterations, each with 1000 particles
smcrf_results <- smcrf(
    method = "smcrf-single-param",
    statistics_target = statistics_target,
    model = model,
    perturb = perturb,
    bounds = bounds,
    parameters_initial = parameters_initial,
    nParticles = c(1000, 1000),
)
# Now we examine the posterior distribution of theta
posterior_iteration <- paste0("Iteration_", (smcrf_results$nIterations + 1))
posterior_theta <- smcrf_results[[posterior_iteration]]$parameters$theta
# We look at the posterior mean of theta
theta_mean <- mean(posterior_theta)
print(theta_mean)
# Notice that the mean is close to 1, the true value of theta.
# We can also look at the posterior variance of theta
theta_var <- var(posterior_theta)
print(theta_var)
# We can also continue the ABC-SMC-RF run if the posterior convergence is not satisfactory
```

```
smcrf_results <- smcrf(
    method = "smcrf-single-param",
    smcrf_results = smcrf_results,
    model = model,
    perturb = perturb,
    bounds = bounds,
    nParticles = c(1000, 1000)
)
# We look again at the posterior mean and variance of theta
posterior_iteration <- paste0("Iteration_", (smcrf_results$nIterations + 1))
posterior_theta <- smcrf_results[[posterior_iteration]]$parameters$theta
theta_mean <- mean(posterior_theta)
print(theta_mean)
theta_var <- var(posterior_theta)
print(theta_var)
# and notice whether there is any improvement in the posterior distribution
# We can continue the runs of ABC-SMC-(D)RF similarly for the examples below
#----------------------------------------------------------------------
#---------------------ABC-SMC-RF for a model with multiple parameters
#----------------------------------------------------------------------
# Data to be fitted consists of two statistics s1 and s2
statistics_target <- data.frame(s1 = 4, s2 = 4)
# We then define a parametrized model for the statistics
model <- function(parameters) {
    statistics <- data.frame(
        s1 = parameters$mu + parameters$theta + runif(nrow(parameters), -0.1, 0.1),
        s2 = parameters$mu * parameters$theta + runif(nrow(parameters), -0.1, 0.1)
    )
    cbind(parameters, statistics)
}
# and a function to perturb the parameters with random noise between iterations
perturb <- function(parameters) {
    if (any(grepl("theta", colnames(parameters)))) {
     parameters[["theta"]] <- parameters[["theta"]] + runif(nrow(parameters), min = -0.1, max = 0.1)
    } else if (any(grepl("mu", colnames(parameters)))) {
     parameters[["mu"]] <- parameters[["mu"]] + runif(nrow(parameters), min = -0.1, max = 0.1)
    }
    return(parameters)
}
# We start from initial guesses from U(-10, 10) x U(-10, 10)
parameters_initial <- data.frame(
    theta = runif(100000, -10, 10),
    mu = runif(100000, -10, 10)
)
# while ensuring that the parameters stay within bounds
bounds <- data.frame(
    parameter = c("theta", "mu"),
    min = c(-10, -10),
    max = c(10, 10)
)
# Finally, we run ABC-SMC-RF with 3 iterations, each with 1000 particles
smcrf_results <- smcrf(
    method = "smcrf-single-param",
    statistics_target = statistics_target,
    model = model,
    perturb = perturb,
    bounds = bounds,
    parameters_initial = parameters_initial,
    nParticles = c(1000, 1000, 1000),
)
# Now we examine the posterior distribution of each parameter
posterior_iteration <- paste0("Iteration_", (smcrf_results$nIterations + 1))
posterior_params <- smcrf_results[[posterior_iteration]]$parameters
posterior_means <- colMeans(posterior_params)
posterior_vars <- var(posterior_params)
print(posterior_means)
print(posterior_vars)
#---------------------------------------------------------------------
#--------------------------------ABC-SMC-DRF for a multivariate model
#---------------------------------------------------------------------
# Data to be fitted consists of 3 statistics s1, s2 and s3
statistics_target <- data.frame(s1 = 9, s2 = 18)
# We then define a parametrized model for the statistics
model <- function(parameters) {
    statistics <- data.frame(
        s1 = parameters$mu + parameters$theta + runif(nrow(parameters), -0.1, 0.1),
        s2 = parameters$mu * parameters$theta + runif(nrow(parameters), -0.1, 0.1)
    )
    cbind(parameters, statistics)
}
# and a function to perturb the parameters with random noise between iterations
perturb <- function(parameters) {
    if (any(grepl("theta", colnames(parameters)))) {
      parameters[["theta"]] <- parameters[["theta"]] + runif(nrow(parameters), min = -0.1, max = 0.1)
    } else if (any(grepl("mu", colnames(parameters)))) {
      parameters[["mu"]] <- parameters[["mu"]] + runif(nrow(parameters), min = -0.1, max = 0.1)
    }
    return(parameters)
}
# We start from initial guesses from U(-10, 10) x U(-10, 10)
theta <- runif(100000, -10, 10)
parameters_initial <- data.frame(
    theta = runif(100000, -10, 10),
    mu = runif(100000, -10, 10)
)
# while ensuring that the parameters stay within bounds
bounds <- data.frame(
    parameter = c("theta", "mu"),
    min = c(-10, -10),
    max = c(10, 10)
)
# Finally, we run ABC-SMC-DRF with 3 iterations, each with 1000 particles
smcrf_results <- smcrf(
    method = "smcrf-multi-param",
    statistics_target = statistics_target,
    model = model,
    perturb = perturb,
    bounds = bounds,
    parameters_initial = parameters_initial,
    nParticles = c(1000, 1000, 1000),
)
# Now we examine the posterior distribution of each parameter
posterior_iteration <- paste0("Iteration_", (smcrf_results$nIterations + 1))
posterior_params <- smcrf_results[[posterior_iteration]]$parameters
```

```
posterior_means <- colMeans(posterior_params)
posterior_vars <- var(posterior_params)
print(posterior_means)
print(posterior_vars)
```

Index